A Component based Testing Technique for a MANET Routing Protocol.

Fatiha Zaïdi, Mounir Lallali, Stephane Maag

A Component based Testing Technique for a MANET Routing Protocol

Fatiha Zadi*† and Mounir Lallali*†
*Univ. Paris-Sud, LRI, UMR 8623, Orsay F-91405, France
†CNRS, Orsay, F-91405, France.
Email: {Zaidi, Lallali}@lri.fr
Stephane Maag‡
‡TELECOM SudParis
CNRS UMR 5157, Evry Cedex, F-91011, France.
Email:Stephane.Maag@it-sudparis.eu

***Abstract*—This paper deals with the crucial challenging issue of testing the conformance of the MANET routing protocols. Indeed, because of the inherent constraints of such networks such as a dynamic topology, to formally test these protocols becomes a tough problem. Most of the studies taking into account a formal model of the protocol is faced to the combinatorial state space explosion issue when deploying and analyzing that model. In our work we present how to cope with that problem by drawing inspiration of the model-checker research domain and an integration of a component-based testing algorithm dedicated to the automatic generation of OLSR test sequences from a formal model written in Promela.**

***Index Terms*—MANET, Routing Protocols, Conformance Testing, Model-checker, SPIN.**

I. Introduction

A wireless mobile ad hoc network (MANET) is a collection of mobile nodes that communicate each other. This kind of network is self-organizing and does not depend on any predefined infrastructure neither centralized administration. By this way, the network may rapidly be deployed. Every node participates to the communication establishment as well reliable operations. Nevertheless, the nodes movement may lead to a volatile topology providing then numerous interconnections. Since the network is infrastructureless, some of the nodes have to behave as routers by interacting using their radio range with open transmission medium in order to establish end-to-end communications. Therefore, due to the inherent constraints of such networks as well the limited resources of the nodes, reliable and efficient routing protocols are needed leading then to the challenging issue of testing the MANET routing protocols.

Many protocols to route the packets through a MANET have now been proposed and several implementations have been provided. Besides, many works proposing testing techniques have been published. However, most of them are related with simulation or emulation testbeds in order to check performance aspects of these protocols. Very few of them take into account their functional properties as well by applying formal approaches. But as established in [1], simulators such as NS-2 may contain erroneous implementations (in their case AODV) on which many studies are based, illustrating by this manner that formal testing techniques are crucial to test such routing protocols.

From that observation, conformance testing techniques that ensure correct protocol implementations have become more and more essential for the development of reliable communicating systems. Due to the complexity to test the systems whose the size becomes huge, the designers use some modularity concepts to model them. Indeed, in many cases and more specifically with the formal model of a MANET whose the topology may be often modified, the analysis of its formal specification may be difficult failing to reach a test objectives and then to generate the test sequences.

That is why component-based testing approach has become a main research area. Its main goal is to generate test sequences for a component embedded in a whole system. The techniques are known for several years now [2], [3] but the constraints occurring when testing a MANET routing protocol lead most of the times to the combinatorial explosion of the model [4]. That is why some efforts have been done to combine techniques provided by both the testing and model-checking communities.

A. Related Works

Although there are many works about testing and verify a MANET [5], from our knowledge, there is no publication on the generation of test sequences from the model of a MANET routing protocol by using model-checking techniques. Nevertheless, we may cite the following works from which we draw inspiration.

For instance, the authors of [6] propose to use the model-checker SPIN [7] in order to automatically generate regression tests. The technique is dedicated to the test of software. Nevertheless, the case study as well the model is very simple and since the model-checker is not modified, the approach is not scalable. In [8], the model-checker Blast is applied [9]. This latter has been modified in order to generate test suites guaranteeing a full coverage of the software. Nevertheless, in this approach no formal specification is performed and it is much more dedicated to the software testing rather than to the routing protocol one. Indeed, it seems difficult with

that technique to observe or control (as needed in a testing architecture) interfaces between some protocol layers.

The same issues are met with the work presented in [10]. The authors apply the Nu-SMV model-checker [11] in order to test the conformance of different kinds of interfaces of Web Services. They use unit and integration testing technique through interfaces between the formal specifications and the data flow. However for a MANET routing protocol, the specification represents several different topologies and to access interfaces embedded in a wireless and mobile node may be difficult. Another interesting work is the one published by Heimdahl et al. [12] where several case studies are analyzed through the application of Nu-SMV. The approach is interesting and original providing relevant results but the model can not contain any variables which significantly reduce the usability in the MANET because of the packet exchanges as well the important flows.

In this paper we cope with these constraints and bridge the existing gap in order to automatically generate test sequences for the conformance of the routing protocols in a MANET.

B. Contributions of this Paper

- While most of the works on MANET routing protocol testing is performed through simulator and emulator, we apply a formal testing approach based especially on the automatic generation of test scenarios from a formal model.
- Taking into account the IETF RFC, we specified a formal model of the OLSR protocol [13] in Promela [14] allowing to provide an OLSR network containing any number of nodes (because of the instantiation of a node process).
- Due to the mobility of the nodes, most of the common testing techniques are irrelevant for the MANET. We draw therefore inspiration from model-checking methods where deployed state compression techniques are reliable.
- We adapt the model-checker SPIN with an efficient generation testing algorithm by developing a tool prototype. Indeed, as mentioned above, several studies tackled that problem but no one applied a component-based testing approach to a model-checker in order to take into account the inherent constraints of a MANET and then avoid the state space explosion issue.
- We automatically generate test sequences for some OLSR test objectives showing by the way that our adaptation is more efficient in some cases than the originating model-checker.

The rest of the paper is organized as follows. In the Section II, we present the main characteristics of OLSR, the concept of component-based conformance testing, the OLSR formal model as well the language utilized to specify the protocol and the SPIN model-checker. Then our test sequence generation approach is illustrated in the Section III. In the Section IV, OLSR test objectives are defined and designed from which test sequences are generated by SPIN and our tool. We comment the results in the Section IV-B before concluding in Section V.

II. OLSR Component-based Testing

This section introduces the protocol OLSR on which our experiments are conducted. We define the concept of component-based conformance testing, present the SPIN model-checker and illustrate the Promela model of OLSR.

A. The Optimized Link State Routing Protocol

The Optimized Link State Routing Protocol (OLSR) [13] is a proactive protocol that needs to react quickly to topology changes and find alternative paths consequently. For that purpose, it maintains several routing tables and exchanges topology information between nodes of the network. OLSR uses specialized aggregation nodes in order to reduce the number of packets used in the flooding of neighborhood information. These nodes are the Multipoint Relay (MPR). They are selected as MPR by some neighbor nodes and they announce periodically the topology information through Topology Control (TC) messages to the nodes which have selected it as MPR. In route calculation the MPR are also used to form the route from a given node to any destination in the network.

B. Component-based Conformance Testing

Conformance testing establishes the conformance of an implementation to its specification. In other words, conformance testing is devoted to the implementation behaves as it is expected by the specification. Commonly, specification is used to generate the tests for observable interactions of the implementation. Those tests are then executed against the real implementation and we observe if the outputs of this one are the expected ones. Accordingly, a verdict is emitted, a *PASS* when the outputs are those expected, a *FAIL* in the other case and an *INCONCLUSIVE* when non determinism is observed.

In case of component-based testing, the accent is put on the interactions of a particular implementation component in the context of the other components. This kind of test is also known as embedded component testing. Indeed, no direct access is given to the component to be tested. To perform component testing, a method is needed to drive the derivation of tests, i.e., to reach in the whole specification the transitions of the component to be tested. In our methodology, the component is characterized by its behavior in terms of interaction with its environment. The behavior of the component is expressed as test objectives to cover.

C. Promela and the Model-Checker SPIN

Promela (PROcess MEta LAnguage) [14] is the modeling language supported by the model-checker tool SPIN. Promela is a high-level description language derived from the C language to specify communication systems. In Promela, system components are described as processes that can interact either by message passing, via channels, or memory sharing via global variables. The communication can be synchronous or asynchronous. Processes specify behavior of the concurrent entities of the distributed system. These processes can be instanciated such that wireless mobile nodes may be obtained

from a single process. Channels and global variables define the environment in which the process run.

SPIN can be used for formal verification of distributed communication systems. The system description is given in Promela and requirements to be verified can be directly written as assertions in the code or by specifying properties as formulas in propositional linear temporal logic PLTL [15]. SPIN works on-the-fly, it means that the tool does not need to construct the complete state space to verify a property; instead it is done dynamically during processing. SPIN provides abstraction techniques such as states fusion, partial order reduction or BDD-like storage techniques.

D. Formal Model of OLSR

To generate the tests, we first need to model in a formal way the protocol. OLSR was specified in Promela which can be viewed as a transition system. This choice has been motivated especially because of the efficient model-checker SPIN and the existing related works. Our specification is devoted to the test of the OLSR implementation embedded in one single node in the context of other OLSR nodes. Besides, we need to take into account the constraints inherent to such networks.

We have to consider the mobility of nodes, the broadcasting of control messages such as HELLO messages and the dynamic modifications of the topology. These aspects raise challenges for the modeling. In our specification, we have made some choices:

- **Broadcast**: In our specification, broadcast is specified by unicasting the packets to every node taking into account a matrix of connectivity. By unicasting we mean that all packets are sent before enabling any other action. In Promela, the messages are sent to all neighbors through channels associated to the nodes.

- **Connectivity**: In our specification, the communications between the nodes are specified in using a dynamic matrix (the number of nodes varies), symmetrical of size $N \times N$ where N is the node number in our network. The matrix is symmetrical because we suppose that all links are bidirectional. This assumption allows to rapidly observe the routes creation and the packets sending as well. Nevertheless, unidirectional links are easily specifiable in using an asymmetric matrix. In addition, in order to reduce the complexity, we suppose that the nodes are connected or not connected (1 or 0 in the matrix). We do not design intermediate state connection as we could have with the protocols of the lower layers (link/MAC). Indeed, our objective is the test of routing functionalities and not the physical or link layer studies. We use a global variable *MaxNodes* that instantiates the maximum number of nodes to be created.

- **Topology**: The topology can change in the specification by applying modification on the connectivity matrix either manually in having beforehand chosen the different moves of the nodes, or randomly.

To cope with a reasonable model, we did some abstractions. First, we do not consider the extensions that are mentioned for IPv6, we only consider IPv4. To simplify, we also do not take into account the HNA (Host Network Association) which is used to have a link to the other networks. We have also simplified the number of interfaces of each node by considering a node with a single interface characterizing its main address. Short pieces of the OLSR specification are illustrated by the Part 1 and Part 2 of Section IV-A.

III. Component Tests Generation and SPIN Model-Checker Algorithms

In this section, we present an embedded test generation method, the Hit-or-Jump (HoJ) algorithm [16] which is integrated into the model-checker SPIN [7]. The integration of our algorithm in the SPIN tool permits to obtain a guided random walk to generate OLSR test sequences. We propose a new algorithm to adapt the acceptance cycle detection algorithm of SPIN by using the HoJ method. Test objectives are used to describe finite behavior and accept state detection algorithm.

A. Outline of the HoJ Algorithm

The HoJ algorithm [16] introduced in SPIN allows to cover all the interactions of the component in its context. The essence of our approach is as follows. At any moment, we conduct a local search from the current state in a neighborhood of the deployed Promela transition system. If an untested part of the component is found (a *Hit*, i.e., an interaction that has to be tested), we keep it for the final test sequence, and then continue the search process from there. Otherwise, we move randomly to the frontier of the neighborhood searched (*Jump*), and resume the process from there. This procedure avoids the building of a whole system accessibility graph. Accordingly, the space required is determined by the user, e.g., a depth limit or a maximum number of states, and it is independent of the system under consideration. On the other hand, a random walk may get trapped at certain part of the component under test [17]. Our algorithm is designed to jump out of the trap and pursue the exploration further. We build at each step a partial accessibility graph to avoid the state-number explosion problem mentioned before. The algorithm finally produces a test sequence as a transition tour of the component in its context.

B. SPIN and HoJ integration

SPIN considers undesirable properties to be checked, which are defined either by Promela *never claims* (i.e., temporal claims) or directly by a PLTL formula. SPIN transforms the claim in a *Bchi* automaton and computes the synchronous product of this automaton with the *Bchi* automaton of the system (i.e., the Promela model). It means that the intersection of these both automata languages is produced providing then a scenario illustrating what should *never* happen.

In our method, the test objectives (i.e., the transitions of the component to be covered) describe a finite desirable behavior. Therefore, by considering the negation of these objectives (a

set of propositions (i.e., boolean expressions) on the system states) and by processing the above mentioned product, we provide a test sequence related to the test objectives.

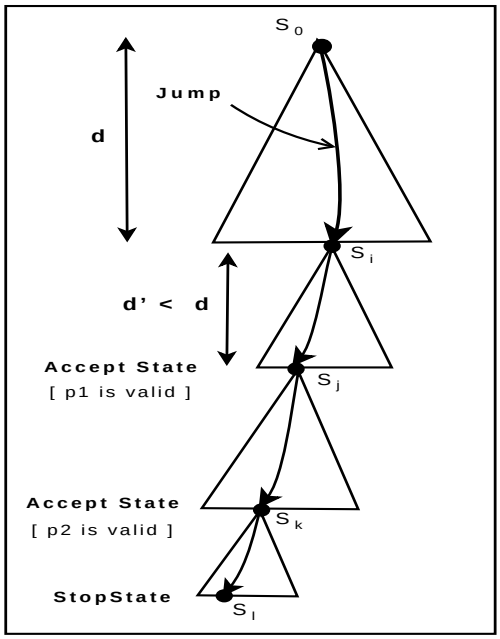
To control the system execution, a synchronous product of the sequence specified in the temporal claim with the interleaving sequences specified in the OLSR Promela model is provided. This synchronous product is a new *Bchi* automaton in which every state is defined as a pair (s, n) where s is a state from the system automaton, and n a state from the *never claim*. Every transition in the new automaton is defined by a pair of transitions (a, p), where a is an action of the system, and p a proposition of the *never claim*. This transition can only occur if the proposition p is valid in s (i.e., the source state of the transition). Note that an *accept state* (s, n) is a state of the product automaton where s is a state that corresponds to an uncovered component transition of the system and n is a state of the claim automaton where a marked proposition (by an *accept label*) is reached. Instead of seeking an acceptance cycle as done in the nested depth-first search (NDFS) algorithm of SPIN, our method search a path (without cycles) going from the initial state of the product automaton to the *stop state* (s, n) where n is a final state of the claim automaton. This path must contains all the *accept states* of the product automaton.

C. Accept State Detection Algorithm

As above mentioned, the main goal is to reach a *stop state* of the synchronous product automaton. During this depth-first search (DFS), all the *accept states* (describing the test objectives) must be reached (cf. Figure 2). If a depth limit d (defined by the user) is reached from a current state (initialized to the initial state in line 1 of Figure 1) without detecting an *accept state* or *stop state*, a jump of depth d is carried out by building a partial search tree and by choosing uniformly and randomly a leaf node of this tree (cf. lines 16–20 in Figure 1). This tree has as root the current state and also has a depth d. The selected leaf node will be the new current state (as in line 21). From this later state, a new partial search (of an *accept state* or *stop state*) is performed (as in line 22). The algorithm terminates when a *stop state* is reached or when the complete exploration of the state space is performed (cf. line 7). Finally, the obtained path from the initial state to the *stop state*, which contains all the *accept states* constitutes the test sequence. The algorithm is depicted in the Figure 1 and illustrated by the Figure 2.

```
1  current_state := initial_state;
2  dfs(current_state);
3
4  proc dfs(s)
5      if not(depth_limit reached) then
6          add(s) to stateSpace;
7          if stop(s) then report testSequence; exit();
8              else if accept(s) then current_state := s; fi
9          fi
10         for each(selected) successor s1 of s do
11             if (s1 not in stateSpace) then
12                 add transition(s,s1) to
                       reachedTransitions;
13                 dfs(s1);
14             fi
15         od
16     else if (s = current_state) then
17         Build from s a partial exploration tree having
               depth limit;
18         Choose uniformly and randomly a leaf node of
               this tree;
19         Initialize the partial stateSpace;
20         Update stateSspace, transitionsTable and
               testSequence;
21         current_state := leafNode
22         dfs(leafNode);
23     fi
24 end dfs.
```

Fig. 1. Accept State Detection Algorithm




```
- Depth Limit = d
- Test Objectives = {p1,p2}
- Test Sequence : S0 ---> Si ---> Sj ---> Sk ---> S1
```

Fig. 2. Accept State Detection with HoJ

D. The Formulation of Test Objectives

The SPIN tool accepts correctness properties expressed in PLTL or directly in Promela *never claim*. In SPIN, the *never claim* must express a negative property. In our component test generation method, we use the *never claim* to describe the test objectives that model the OLSR component transitions to be tested. These test objectives describe finite desirable behaviors. Their formulation consists of a set of logical propositions on the model state which can correspond to the triggering of transitions or to the evaluation of variables. Every proposition is marked by an *accept label*. We give below an example of a *never claim* with one *accept label* (i.e., *accept_prop1* in line 6 of the following claim) that identifies where the test objective holds. The *never claim* is noted $\neg(\diamond\, prop1)$ in PLTL. In PROMELA, it is noted as in the following:

```
never {
    do
    :: skip
    :: prop1 −> break
    od
    accept_prop1: skip
}
```

The loop is left when all the propositions (i.e., test objectives) are checked in order to finally reach the final state of the *never claim*. For our experiments on OLSR (see Section IV), we need to define a *never claim* with several *accept label* that mark a set of propositions.

IV. Experimental Results

Our experiments have been performed with a Processor Intel Pentium 4 (CPU 2.4GHz, RAM 512MB, Cache size 512 MB). The OLSR test sequences have been generated both by SPIN and our prototype tool based on the model-checker in which HoJ has been integrated, allowing then to compare the results.

A. OLSR Test Objectives

This section presents some examples of test objectives formulation. For sake of simplicity, we only detail four OLSR properties. The experiments are performed on a network modeled with five nodes moving randomly. We nevertheless assume that this random topology allows to observe the following properties especially to generate the respected test sequences. We formalize the test objectives for the node $n1$.

- **Test Objective 1:** To test the asymmetric link process. A node $n1$ sends a HELLO message to its neighbors. Then, it can declare that it has an asymmetric link with a node if this node sends back a HELLO message with the source $n1$ added in its ASYM set.
- **Test Objective 2:** To test the symmetrical link process. A node $n1$ which has already a asymmetrical link with a node $n2$, can declare to have a symmetrical link with $n2$ if it sends a HELLO message saying to $n1$ that it has a symmetrical link with him.
- **Test Objective 3:** To test that only a node that is established as a MPR can send TC messages (see Figure 3).
- **Test Objective 4:** To test that a HELLO message is sent after a Hello interval.

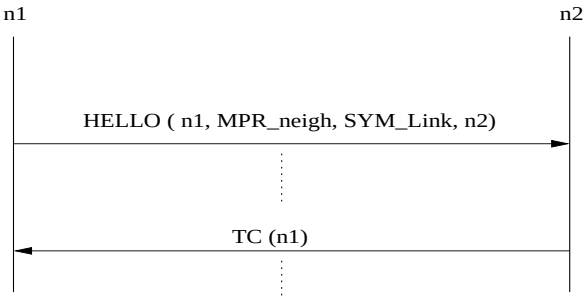


Fig. 3. MSC of the test objective 3

To perform our test generation, we express these test objectives inside the *never claim*. We identify in the formal specification, the component involved in the design of the test objectives. For that purpose, we label in the specification the transitions to be reached. These transitions will belong to the test sequence.

In order to generate the test sequence for the test objective 1, we put two (control-flow) labels in the specification, one in the HELLO message generation part (cf. label $L0$ in line 4 of the Part 1) and a second in the part devoted to the classification of the links (cf. label $L1$ line 8 of the Part 2). We show below how the labels $L0$ and $L1$ are put on the Promela specification.

```
1  do
2  ::( i<NODES)−> if
3      :: (Node_1[id].Node_2[i] == UNSPEC_LINK) −> skip;
4      :: else −> L0 : Ch[i] ! HELLO_MESSAGE,sendMe;
5      fi ; i++;
6  :: else −> break;
7  od;
```

Part 1: Promela Specification of HELLO Message Generation with the Label L0

```
1  inline calculLinkType(indice, linkCode_LT) {
2      if
3      :: ((LinkSet[indice].L_SYM_time.sec > timer.sec) ||
           ((LinkSet[indice].L_SYM_time.sec == timer.sec)
           && (LinkSet[indice].L_SYM_time.msec >=
           timer.msec)))
4          −> linkCode_LT = SYM_LINK;
5      :: else −>
6          if
7          :: ((LinkSet[indice].L_ASYM_time.sec > timer.sec)
                || ((LinkSet[indice].L_ASYM_time.sec ==
               timer.sec) &&
               (LinkSet[indice].L_ASYM_time.msec >=
               timer.msec)))
8              −> linkCode_LT = ASYM_LINK; L1 : skip;
9          :: else −> linkCode_LT = LOST_LINK;
10         fi ;
11     fi ;
12 }
```

Part 2: Promela Specification of Links Classification with the Label L1

In this test objective 1, first a HELLO message has to be sent from $n1$ to $n2$ (cf. label $L0$ of the Part 1), then $n2$ receives it, answers and $n1$ classifies its link to this node as asymmetric (cf. label $L1$ of the Part 2). Two steps are needed in the *never claim* and two global variables are defined as it follows:

```
#define p1 (Node[1]@L0)
#define p2 (Node[1]@L1)
```

Note that the variable $p1$, for instance, takes the value $true$ if its associated process instance ($Node[1]$) is currently in the state marked by $L0$ which is the first step of the test objective 1. Let $checked$ be an array of boolean indexed by the boolean variables $p1, p2$ such that $checked(p1) = true$ meaning as an example that the variable $p1$ takes $true$ at least once. These boolean variables $p1, p2$ are used in the design of the test objective 1. Its associated *never claim* is described by the two propositions: $prop1 = p1 \wedge \neg checked(p1)$; $prop2 = p2 \wedge \neg checked(p2) \wedge checked(p1)$. The never claim of the test objective 1 is described as follows:

```
never {
    do
    :: skip
    :: prop1 -> checked(p1) = true;
    :: prop2 -> checked(p2) = true; break;
    od
    accept_objective1 : skip
}
```

The never claim of the test objective 1

B. Results and Discussions

This section presents a comparative table (see Figure I) between the results obtained by the original SPIN and by our own prototype for the OLSR protocol in the same context. We have exercised our method to test a specific node in the context of four other nodes. The test objectives are expressed in order to cover some OLSR properties. We compared the stored states number (in data structures) and the stored or matched transitions number.

We note that for the two tools (SPIN and our own prototype), we obtained the same test sequence with the same depth. We did not try to obtain different test sequences by changing the depth of search. Our objective is to achieve to reach the transitions in the OLSR Promela model. We can notice that for the MPR with the SPIN tool, we did not achieve to reach the test objective related to the computation of the MPR. Nevertheless, with our prototype we obtain a result for this test objective. We can explain these results by our efficient strategy to explore the system. Indeed, we can get out a part of the specification that is not relevant for our search with the jump strategy. SPIN DFS carries out a back-tracking if the maximum search depth is reached without reaching an *accept state* or a *stop state*. Indeed, a maximal depth limit is predefined and the value is high.

On the contrary, our prototype works on partial graph and traverses shortest path due to the small depth limit set to obtain a jump. In this case we only back-track to the root node of the partial graph (i.e., current state) and executes a jump. Moreover, the jump allows us to get out the *trap* and to search our test objectives in other part of the graph. We can avoid very long test sequences with uninteresting paths according to a good depth search choice. To explain more precisely the MPR test objective, we have to say that the other test objectives are related to a subset of nodes as these properties are computed locally for a node. On contrary, for the property of MPR, the computation are done for all the nodes of the network, there is no knowledge a priori of which node can be computed as MPR. The flooding of HELLO messages and the classification of links are performed for all the neighborhood. This computation can explain the state space explosion even if the network is reduced to five nodes. With our modified version of SPIN, we obtain a result after a *jump*. In that work, we generated the test sequences related to OLSR properties. Since they were provided from a Promela model, they are represented as input/outputs sequences. Therefore, a following work is to formulate them in a executable manner and to apply them on a real implementation under test of OLSR as it is mentioned in a previous work [18] in which we define testing architectures.

V. Conclusion

This paper discussed a new approach to generate test cases for an embedded component, which is a MANET routing protocol. This method conducts a search in the partial product of the whole formally specified OLSR system and the automaton of the formula that represents the test objectives, i.e., the transitions of the component to cover.

We have implemented inside the model-checker SPIN an efficient testing algorithm that allows to avoid the state space explosion. Therefore, through an OLSR specification written in Promela, we have exercised our prototype against SPIN. While the results are quite similar if the component to reach is not so deep into the model (that is, it does not need an important testing coverage), they differ when many interactions between the nodes are needed to observe a test objective. Indeed, our approach may take into account the dynamic topology of MANET networks and then provides test verdicts whereas this is not the case for the model-checker.

An immediate line of future is to exercise our prototype with different criteria of coverage. We present in the paper a coverage criterion that is the transitions coverage in order to produce a transitions tour of the component in its context. This criterion can be rapidly changed, may one want only to test some critical functionalities of a module. Specifically, it would help to test the scalability of our approach with dozen of nodes. Besides, it will be interesting to validate our approach through complex OLSR testing objectives. For that aspect, requirements such as HNA and two OLSR interfaces should be added to the formal model.

References

[1] K. Bhargavan, C. Gunter, I. Lee, O. Sokolsky, M. Kim, D. Obradovic, and M. Viswanathan, "Verisim: Formal analysis of network simulations," *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, pp. 129–145, 2002.

[2] C. Bourhfir, R. Dssouli, E. Aboulhamid, and N. Rico, "A guided incremental test case generation procedure for conformance testing for CEFSM specified protocols," in *IWTCS'98*, Tomsk, Russia, 1998.

[3] A. Cavalli, B. Defude, C. Rinderknecht, and F. Zadi, "A service-component testing method and suitable CORBA architecture," in *Proceedings of the Sixth IEEE ISCC'01*, Tunisia, 2001, pp. 655–660.

[4] S. Maag and F. Zadi, "Testing methodology for an ad hoc routing protocol," in *Proc. of First ACM Workshop PM2HW2N*, Torremolinos, (Malaga), Spain, 2006.

[5] A. C. Viana, S. Maag, and F. Zaidi, "One step forward: Linking wireless self-organising networks validation techniques with formal testing approaches," *ACM Survey*, vol. 41, no. 3, 2009.

[6] L. Xu, M. Dias, and D. Richardson, "Generating regression tests via model checking," in *Proceedings of COMPSAC'04*, 2004.

[7] G. J. Holzmann, *The SPIN Model Checker - Primer and Reference Manual*. Addison-Wesley, Reading Massachusetts, 2004, 608 pgs.

[8] D. Beyer, A. Chlipala, T. Henzinger, R. Jhala, and R. Majumdar, "Generating tests from counterexamples," in *Proceedings of ICSE'04*, 2004.

[9] D. Beyer, T. Henzinger, R. Jhala, and R. Majumdar, "The software model checker blast: Applications to software engineering." *Int. Journal on Software Tools for Technology Transfer*, 2007.

[10] Y. Zheng, J. Zhou, and P. Krause, "A model checking based test case generation framework forweb services," in *Proceedings of ITNG'07*, 2007.

Invariant	Violation Depth		Stored States		Stored + Matched Transitions		Duration (s)	
	Spin	**Prototype**	**Spin**	**Prototype**	**Spin**	**Prototype**	**Spin**	**Prototype**
Asymmetric link	732	732	328	328	328	328	3.31	3.28
Symmetric link	932	932	428	428	428	428	3.33	3.31
HELLO_INTERVAL	691	691	310	310	310	310	3.30	3.28
MPR Election	-	2022	-	905	-	905	-	7.46

TABLE I
DISCUSSIONS OF RESULTS

[11] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "Nusmv: a new symbolic model verifier," in *Proceedings of CAV'99*. Trento, Italy: Springer, 1999, pp. 495–499.

[12] S. Rayadurgam and M. Heimdahl, "Generating mc/dc adequate test sequences through model checking," in *Proceedings of SEW'03 Workshop*, 2003.

[13] T.Clausen and P.Jacquet, *Optimized Link State Routing Protocol (OLSR) - RFC3626*, ietf ed., INRIA, 2003.

[14] R. Gerth, "Concise Promela Reference," June 1997, http://www.spinroot.com/spin/Man/Quick.html.

[15] A. Pnueli, "The temporal logic of programs," *Proc. 18th IEEE Symposium on Foundations of Computer Science*, pp. 46–57, 1977.

[16] A. Cavalli, D. Lee, C. Rinderknecht, and F. Zadi, "Hit-or-Jump : An algorithm for embedded testing with applications to IN services," in *Proceedings of FORTE/PSTV'99*, Beijing, China, 1999.

[17] D. Lee, K. Sabnani, D. Kristol, and S. Paul, "Conformance Testing of Protocols Specified as Communicating Finite State Machines - A Guided Random Walk Based Approach," in *IEEE Transactions on Communications*, vol. 44, No.5, 1996.

[18] S. Maag, C. Grepet, and A. Cavalli, "A formal validation methodology for manet routing protocols based on nodes' self similarity," *Computer Communications*, vol. 31, no. 4, pp. 827–841, 2008.